1 Purpose

nag_dggsvp (f08vec) uses orthogonal transformations to simultaneously reduce the \( m \) by \( n \) matrix \( A \) and the \( p \) by \( n \) matrix \( B \) to upper triangular form. This factorization is usually used as a preprocessing step for computing the generalized singular value decomposition (GSVD).

2 Specification

```c
#include <nag.h>
#include <nagf08.h>

void nag_dggsvp (Nag_OrderType order, Nag_ComputeUType jobu,
    Nag_ComputeVType jobv, Nag_ComputeQType jobq, Integer m, Integer p,
    Integer n, double a[], Integer pda, double b[], Integer pdb,
    double tola, double tolb, Integer *k, Integer *l, double u[],
    Integer pdu, double v[], Integer pdv, double q[], Integer pdq,
    NagError *fail)
```

3 Description

nag_dggsvp (f08vec) computes orthogonal matrices \( U \), \( V \) and \( Q \) such that

\[
U^T AQ = \begin{cases} 
  \begin{pmatrix} 
    n-k-l & k & l \\
    k & 0 & \lambda_{12} & \lambda_{13} \\
    l & 0 & 0 & \lambda_{23} \\
    m-k-l & 0 & 0 & 0 
  \end{pmatrix}, & \text{if } m-k-l \geq 0; \\
  \begin{pmatrix} 
    n-k-l & k & l \\
    k & 0 & \lambda_{12} & \lambda_{13} \\
    m-k-l & 0 & 0 & \lambda_{23} 
  \end{pmatrix}, & \text{if } m-k-l < 0; 
\end{cases}
\]

\[
V^T BQ = \begin{pmatrix} 
  l \\
  p-l \\
\end{pmatrix} \begin{pmatrix} 
  n-k-l & k & l \\
  0 & 0 & \lambda_{13} \\
  0 & 0 & 0 
\end{pmatrix}
\]

where the \( k \) by \( k \) matrix \( \lambda_{12} \) and \( l \) by \( l \) matrix \( \lambda_{13} \) are nonsingular upper triangular; \( \lambda_{23} \) is \( l \) by \( l \) upper triangular if \( m-k-l \geq 0 \) and is \( (m-k) \) by \( l \) upper trapezoidal otherwise. \( (k+l) \) is the effective numerical rank of the \( (m+p) \) by \( n \) matrix \( (A^T B^T)^T \).

This decomposition is usually used as the preprocessing step for computing the Generalized Singular Value Decomposition (GSVD), see function nag_dggsvd (f08vac).
5 Arguments

1: order – Nag_OrderType
   
   On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: jobu – Nag_ComputeUType
   
   On entry: if jobu = Nag_AllU, the orthogonal matrix U is computed.
   If jobu = Nag_NotU, U is not computed.

   Constraint: jobu = Nag_AllU or Nag_NotU.

3: jobv – Nag_ComputeVType
   
   On entry: if jobv = Nag_ComputeV, the orthogonal matrix V is computed.
   If jobv = Nag_NotV, V is not computed.

   Constraint: jobv = Nag_ComputeV or Nag_NotV.

4: jobq – Nag_ComputeQType
   
   On entry: if jobq = Nag_ComputeQ, the orthogonal matrix Q is computed.
   If jobq = Nag_NotQ, Q is not computed.

   Constraint: jobq = Nag_ComputeQ or Nag_NotQ.

5: m – Integer
   
   On entry: m, the number of rows of the matrix A.

   Constraint: m ≥ 0.

6: p – Integer
   
   On entry: p, the number of rows of the matrix B.

   Constraint: p ≥ 0.

7: n – Integer
   
   On entry: n, the number of columns of the matrices A and B.

   Constraint: n ≥ 0.

8: a[dim] – double
   
   Note: the dimension, dim, of the array a must be at least
   max(1, pda × n) when order = Nag_ColMajor;
   max(1, m × pda) when order = Nag_RowMajor.
The \((i, j)\)th element of the matrix \(A\) is stored in
\[
\begin{align*}
  &\text{for } \text{order} = \text{Nag\_ColMajor}:
  &a[(j - 1) \times pda + i - 1], \\
  &\text{for } \text{order} = \text{Nag\_RowMajor}:
  &a[(i - 1) \times pda + j - 1].
\end{align*}
\]

On entry: the \(m\) by \(n\) matrix \(A\).

On exit: contains the triangular (or trapezoidal) matrix described in Section 3.

9: \textbf{pda} – Integer

\textit{Input}

On entry: the stride separating row or column elements (depending on the value of \textit{order}) in the array \(a\).

Constraints:

\[
\begin{align*}
  &\text{if } \text{order} = \text{Nag\_ColMajor}, \ pda \geq \max(1, m); \\
  &\text{if } \text{order} = \text{Nag\_RowMajor}, \ pda \geq \max(1, n).
\end{align*}
\]

10: \textbf{b[\text{dim}]} – double

\textit{Input/Output}

Note: the dimension, \textit{dim}, of the array \(b\) must be at least
\[
\begin{align*}
  &\text{for } \text{order} = \text{Nag\_ColMajor}, \ \max(1, \text{pdb} \times n); \\
  &\text{for } \text{order} = \text{Nag\_RowMajor}, \ \max(1, p \times \text{pdb}).
\end{align*}
\]

The \((i, j)\)th element of the matrix \(B\) is stored in
\[
\begin{align*}
  &\text{for } \text{order} = \text{Nag\_ColMajor}:
  &b[(j - 1) \times \text{pdb} + i - 1]; \\
  &\text{for } \text{order} = \text{Nag\_RowMajor}:
  &b[(i - 1) \times \text{pdb} + j - 1].
\end{align*}
\]

On entry: the \(p\) by \(n\) matrix \(B\).

On exit: contains the triangular matrix described in Section 3.

11: \textbf{pdb} – Integer

\textit{Input}

On entry: the stride separating row or column elements (depending on the value of \textit{order}) in the array \(b\).

Constraints:

\[
\begin{align*}
  &\text{if } \text{order} = \text{Nag\_ColMajor}, \ \text{pdb} \geq \max(1, p); \\
  &\text{if } \text{order} = \text{Nag\_RowMajor}, \ \text{pdb} \geq \max(1, n).
\end{align*}
\]

12: \textbf{tola} – double

13: \textbf{tolb} – double

\textit{Input}

On entry: \textit{tola} and \textit{tolb} are the thresholds to determine the effective numerical rank of matrix \(B\) and a subblock of \(A\). Generally, they are set to
\[
\begin{align*}
  &\text{tola} = \max(m, n)\|A\|_\infty \epsilon, \\
  &\text{tolb} = \max(p, n)\|B\|_\infty \epsilon,
\end{align*}
\]

where \(\epsilon\) is the \textit{machine precision}.

The size of \textit{tola} and \textit{tolb} may affect the size of backward errors of the decomposition.

14: \textbf{k} – Integer *

15: \textbf{l} – Integer *

\textit{Output}

On exit: \textit{k} and \textit{l} specify the dimension of the subblocks \(k\) and \(l\) as described in Section 3; \((k + l)\) is the effective numerical rank of \(\begin{pmatrix} a^\top & b^\top \end{pmatrix}^\top\).
16: $u[dim]$ – double

**Output**

**Note:** the dimension, $dim$, of the array $u$ must be at least

\[
\max(1, pdu \times m) \quad \text{when } jobu = \text{Nag AllU};
\]

1 otherwise.

The $(i,j)$th element of the matrix $U$ is stored in

\[
\begin{align*}
&u[(j - 1) \times pdu + i - 1] \quad \text{when } order = \text{Nag ColMajor}; \\
&q[(i - 1) \times pdu + j - 1] \quad \text{when } order = \text{Nag RowMajor}.
\end{align*}
\]

*On exit:* if $jobu = \text{Nag AllU}$, $u$ contains the orthogonal matrix $U$.

If $jobu = \text{Nag NotU}$, $u$ is not referenced.

17: $pdu$ – Integer

**Input**

*On entry:* the stride separating row or column elements (depending on the value of $order$) in the array $u$.

**Constraints:**

\[
\begin{align*}
&\text{if } jobu = \text{Nag AllU}, \quad pdu \geq \max(1, m); \\
&\text{otherwise } pdu \geq 1.
\end{align*}
\]

18: $v[dim]$ – double

**Output**

**Note:** the dimension, $dim$, of the array $v$ must be at least

\[
\max(1, pdv \times p) \quad \text{when } jobv = \text{Nag ComputeV};
\]

1 otherwise.

The $(i,j)$th element of the matrix $V$ is stored in

\[
\begin{align*}
&v[(j - 1) \times pdv + i - 1] \quad \text{when } order = \text{Nag ColMajor}; \\
&q[(i - 1) \times pdv + j - 1] \quad \text{when } order = \text{Nag RowMajor}.
\end{align*}
\]

*On exit:* if $jobv = \text{Nag ComputeV}$, $v$ contains the orthogonal matrix $V$.

If $jobv = \text{Nag NotV}$, $v$ is not referenced.

19: $pdv$ – Integer

**Input**

*On entry:* the stride separating row or column elements (depending on the value of $order$) in the array $v$.

**Constraints:**

\[
\begin{align*}
&\text{if } jobv = \text{Nag ComputeV}, \quad pdv \geq \max(1, p); \\
&\text{otherwise } pdv \geq 1.
\end{align*}
\]

20: $q[dim]$ – double

**Output**

**Note:** the dimension, $dim$, of the array $q$ must be at least

\[
\max(1, pdq \times n) \quad \text{when } jobq = \text{Nag ComputeQ};
\]

1 otherwise.

The $(i,j)$th element of the matrix $Q$ is stored in

\[
\begin{align*}
&q[(j - 1) \times pdq + i - 1] \quad \text{when } order = \text{Nag ColMajor}; \\
&q[(i - 1) \times pdq + j - 1] \quad \text{when } order = \text{Nag RowMajor}.
\end{align*}
\]

*On exit:* if $jobq = \text{Nag ComputeQ}$, $q$ contains the orthogonal matrix $Q$.

If $jobq = \text{Nag NotQ}$, $q$ is not referenced.
21:  pdq – Integer  

   Input

   On entry: the stride separating row or column elements (depending on the value of order) in the array q.

   Constraints:

   if jobq = Nag_ComputeQ, pdq ≥ max(1, n);
   otherwise pdq ≥ 1.

22:  fail – NagError*

   Input/Output

   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

   Dynamic memory allocation failed.

   See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

   On entry, argument ⟨value⟩ had an illegal value.

NE_ENUM_INT_2

   On entry, jobq = ⟨value⟩, pdq = ⟨value⟩ and n = ⟨value⟩.

   Constraint: if jobq = Nag_ComputeQ, pdq ≥ max(1, n);
   otherwise pdq ≥ 1.

   On entry, jobu = ⟨value⟩, pdu = ⟨value⟩ and m = ⟨value⟩.

   Constraint: if jobu = Nag_AllU, pdu ≥ max(1, m);
   otherwise pdu ≥ 1.

   On entry, jobv = ⟨value⟩, pdv = ⟨value⟩ and p = ⟨value⟩.

   Constraint: if jobv = Nag_ComputeV, pdv ≥ max(1, p);
   otherwise pdv ≥ 1.

NE_INT

   On entry, m = ⟨value⟩.

   Constraint: m ≥ 0.

   On entry, n = ⟨value⟩.

   Constraint: n ≥ 0.

   On entry, p = ⟨value⟩.

   Constraint: p ≥ 0.

   On entry, pda = ⟨value⟩.

   Constraint: pda > 0.

   On entry, pdb = ⟨value⟩.

   Constraint: pdb > 0.

   On entry, pdq = ⟨value⟩.

   Constraint: pdq > 0.

   On entry, pdu = ⟨value⟩.

   Constraint: pdu > 0.

   On entry, pdv = ⟨value⟩.

   Constraint: pdv > 0.
NE_INT_2

On entry, $\text{pda} = \langle\text{value}\rangle$ and $m = \langle\text{value}\rangle$.
Constraint: $\text{pda} \geq \max(1, m)$.

On entry, $\text{pda} = \langle\text{value}\rangle$ and $n = \langle\text{value}\rangle$.
Constraint: $\text{pda} \geq \max(1, n)$.

On entry, $\text{pdb} = \langle\text{value}\rangle$ and $n = \langle\text{value}\rangle$.
Constraint: $\text{pdb} \geq \max(1, n)$.

On entry, $\text{pdb} = \langle\text{value}\rangle$ and $p = \langle\text{value}\rangle$.
Constraint: $\text{pdb} \geq \max(1, p)$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

The computed factorization is nearly the exact factorization for nearby matrices $(A + E)$ and $(B + F)$, where

$$
\|E\|_2 = O(\epsilon)\|A\|_2 \quad \text{and} \quad \|F\|_2 = O(\epsilon)\|B\|_2,
$$

and $\epsilon$ is the machine precision.

8 Parallelism and Performance

nag_dggsvp (f08vec) is not threaded by NAG in any implementation.

nag_dggsvp (f08vec) makes calls to BLAS and/or LAPACK routines, which may be threaded within the
vendor library used by this implementation. Consult the documentation for the vendor library for further
information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the
OpenMP environment used within this function. Please also consult the Users’ Note for your
implementation for any additional implementation-specific information.

9 Further Comments

The complex analogue of this function is nag_zggsvp (f08vsc).

10 Example

This example finds the generalized factorization

$$
A = U\Sigma_1 (0 \quad S) Q^T, \quad B = V\Sigma_2 (0 \quad T) Q^T,
$$

of the matrix pair $(A \quad B)$, where

$$
A = \begin{pmatrix}
1 & 2 & 3 \\
3 & 2 & 1 \\
4 & 5 & 6 \\
7 & 8 & 8
\end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix}
-2 & -3 & 3 \\
4 & 6 & 5
\end{pmatrix}.
$$
10.1 Program Text

/* nag_dggsvp (f08vec) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double eps, norm, tola, tolb;
    Integer i, irank, j, k, i, m, n, nrows, p, pda, pdb, pdq, pdu, pdv;
    Integer exit_status = 0;
    /* Arrays */
    double *a = 0, *b = 0, *q = 0, *u = 0, *v = 0;
    /* Nag Types */
    NagError fail;
    Nag_OrderType order;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J-1)*pda +I-1 ]
    #define B(I, J) b[(J-1)*pdb +I-1 ]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda +J-1 ]
    #define B(I, J) b[(I-1)*pdb +J-1 ]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    printf("nag_dggsvp (f08vec) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n"]);
    #else
    scanf("%*[\n"]);
    #endif
    #ifdef _WIN32
    scanf_s("%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n"", &m, &n, &p);
    #else
    scanf("%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n"", &m, &n, &p);
    #endif
    if (n < 0 || m < 0 || p < 0)
    {
        printf("Invalid n, m or p\n");
        exit_status = 1;
        goto END;
    }

    #ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdb = p;
    pdv = p;
    #else
    pda = n;
    pdb = n;
    pdv = m;
    #endif

    INIT_FAIL(fail);
    printf("nag_dggsvp (f08vec) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n"]);
    #else
    scanf("%*[\n"]);
    #endif
    #ifdef _WIN32
    scanf_s("%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n"", &m, &n, &p);
    #else
    scanf("%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n"", &m, &n, &p);
    #endif
    if (n < 0 || m < 0 || p < 0)
    {
        printf("Invalid n, m or p\n");
        exit_status = 1;
        goto END;
    }

    #ifdef NAG_COLUMN_MAJOR
    pda = m;
    pdb = p;
    pdv = p;
    #else
    pda = n;
    pdb = n;
    pdv = m;
    #endif
/* Allocate memory */
if (!(a = NAG_ALLOC(m*n, double)) ||
    !(b = NAG_ALLOC(p*n, double)) ||
    !(q = NAG_ALLOC(n*n, double)) ||
    !(u = NAG_ALLOC(m*m, double)) ||
    !(v = NAG_ALLOC(p*m, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read the m by n matrix A and p by n matrix B from data file */
for (i = 1; i <= m; ++i)
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
    scanf("%*[\n]");
for (i = 1; i <= p; ++i)
    for (j = 1; j <= n; ++j) scanf("%lf", &B(i, j));
    scanf("%*[\n]");
/* get norms of A and B using nag_dge_norm (f16rac). */
    nag_dge_norm(order, Nag_OneNorm, m, n, a, pda, &norm, &fail);
    nag_dge_norm(order, Nag_OneNorm, p, n, b, pdb, &norm, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Get the machine precision, using nag_machine_precision (x02ajc) */
eps = nag_machine_precision;
tola = MAX(m, n) * norm * eps;
tolb = MAX(p, n) * norm * eps;
/* Compute the factorization of (A, B) (A = U*S*(Q**T), B = V*T*(Q**T)) */
    nag_dggsvp(order, Nag_AllU, Nag_ComputeV, Nag_ComputeQ, m, p, n, a, pda, b, pdb, tola, tolb, &k, &l, u, pdu, v, pdv, q, pdq, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dggsvp (f08vec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print details of the generalized SVD */
irank = k + l;
printf("Numerical rank of (A**T B**T)**T (K+L)\n\n"NAG_IFMT"\n\n", irank);
nrows = MIN(m,irank);
flush(stdout);
nag_gen_real_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, nrows,
irank, &A(1, n - irank + 1), pda, "%13.4e",
"Matrix S", Nag_IntegerLabels, 0,
Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR) goto FAIL;
fflush(stdout);
nag_gen_real_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, l, l,
&B(1, n - l + 1), pdb, "%13.4e",
"Upper triangular matrix T", Nag_IntegerLabels,
0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR) goto FAIL;
printf("\n");
fflush(stdout);
nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
m, u, pdu, "%13.4e", "Orthogonal matrix U",
Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
80, 0, 0, &fail);
if (fail.code != NE_NOERROR) goto FAIL;
fflush(stdout);
nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, p, p,
v, pdv, "%13.4e", "Orthogonal matrix V",
Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
0, 0, &fail);
if (fail.code != NE_NOERROR) goto FAIL;
fflush(stdout);
nag_gen_real_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
q, pdq, "%13.4e", "Orthogonal matrix Q",
Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80,
0, 0, &fail);
FAIL:
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print_comp (x04cbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(q);
NAG_FREE(u);
NAG_FREE(v);

return exit_status;
}

10.2 Program Data

nag_dggsvp (f08vec) Example Program Data

4 3 2 : m, n and p
1.0 2.0 3.0
3.0 2.0 1.0
4.0 5.0 6.0
7.0 8.0 8.0 : matrix A
-2.0 -3.0 3.0
4.0 6.0 5.0 : matrix B
### 10.3 Program Results

nag_dggsvp (f08vec) Example Program Results

Numerical rank of $(A^{**T} B^{**T})^{**T} (K+L)$

3

Matrix S

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.0569e+00</td>
<td>1.0771e+01</td>
<td>-7.2814e+00</td>
</tr>
<tr>
<td>2</td>
<td>7.1947e+00</td>
<td>-7.5262e+00</td>
<td>5.8129e-01</td>
</tr>
</tbody>
</table>

Upper triangular matrix T

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.0623e+00</td>
<td>-3.1305e+00</td>
</tr>
<tr>
<td>2</td>
<td>-4.9193e+00</td>
<td></td>
</tr>
</tbody>
</table>

Orthogonal matrix U

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.3484e-01</td>
<td>5.1025e-01</td>
<td>-2.4351e-01</td>
<td>8.1373e-01</td>
</tr>
<tr>
<td>2</td>
<td>6.7420e-01</td>
<td>-5.4670e-01</td>
<td>3.5349e-01</td>
<td>3.4874e-01</td>
</tr>
<tr>
<td>3</td>
<td>2.6968e-01</td>
<td>4.8292e-01</td>
<td>-6.9127e-01</td>
<td>-4.6499e-01</td>
</tr>
<tr>
<td>4</td>
<td>6.7420e-01</td>
<td>4.5558e-01</td>
<td>5.8129e-01</td>
<td>1.5127e-15</td>
</tr>
</tbody>
</table>

Orthogonal matrix V

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4.4721e-01</td>
<td>8.9443e-01</td>
</tr>
<tr>
<td>2</td>
<td>8.9443e-01</td>
<td>4.4721e-01</td>
</tr>
</tbody>
</table>

Orthogonal matrix Q

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-8.3205e-01</td>
<td>5.5470e-01</td>
<td>0.0000e+00</td>
</tr>
<tr>
<td>2</td>
<td>5.5470e-01</td>
<td>8.3205e-01</td>
<td>0.0000e+00</td>
</tr>
<tr>
<td>3</td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>-1.0000e+00</td>
</tr>
</tbody>
</table>